Introduction to time series and stationarity

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

Course content

You will learn

- Structure of ARIMA models
- How to fit ARIMA model
- How to optimize the model
- How to make forecasts
- How to calculate uncertainty in predictions

Loading and plotting

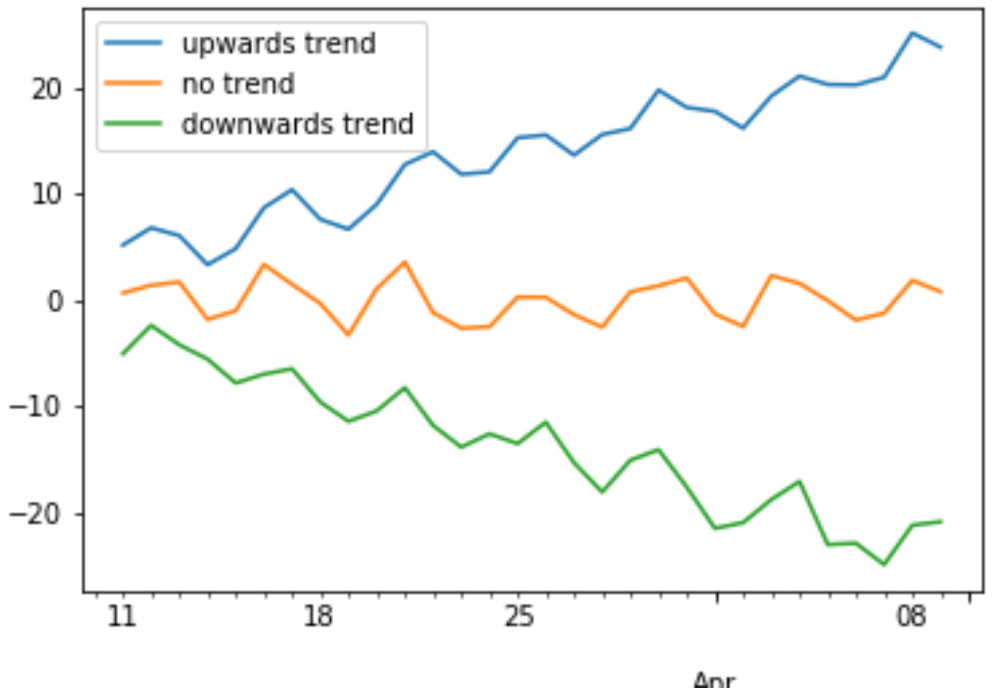
```
import pandas as pd
import matplotlib as plt

df = pd.read_csv('time_series.csv', index_col='date', parse_dates=True)
```

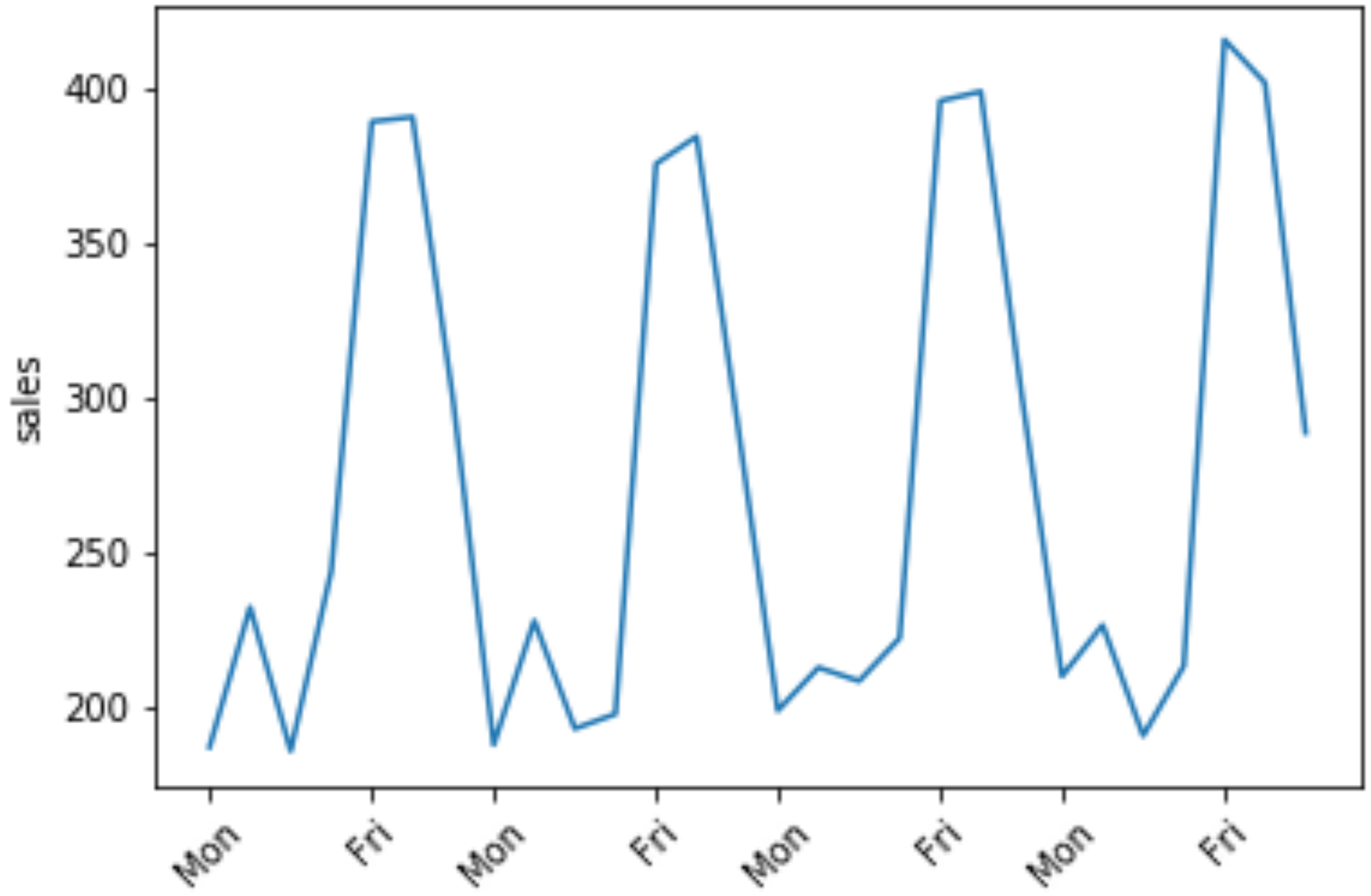
```
date            values
2019-03-11    5.734193
2019-03-12    6.288708
2019-03-13    5.205788
2019-03-14    3.176578
```

Trend

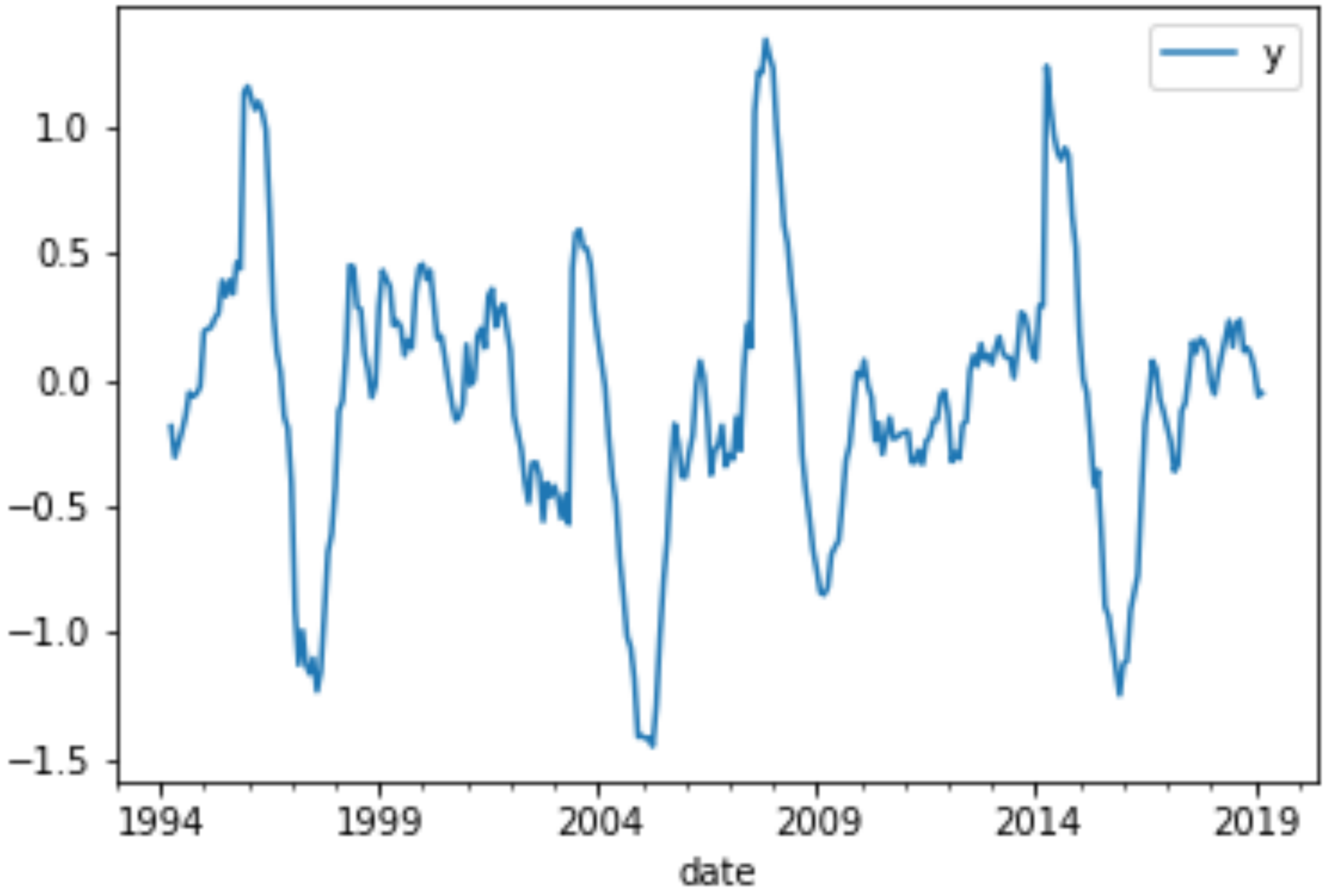
```
fig, ax = plt.subplots()
df.plot(ax=ax)
plt.show()
```



Seasonality



Cyclicality



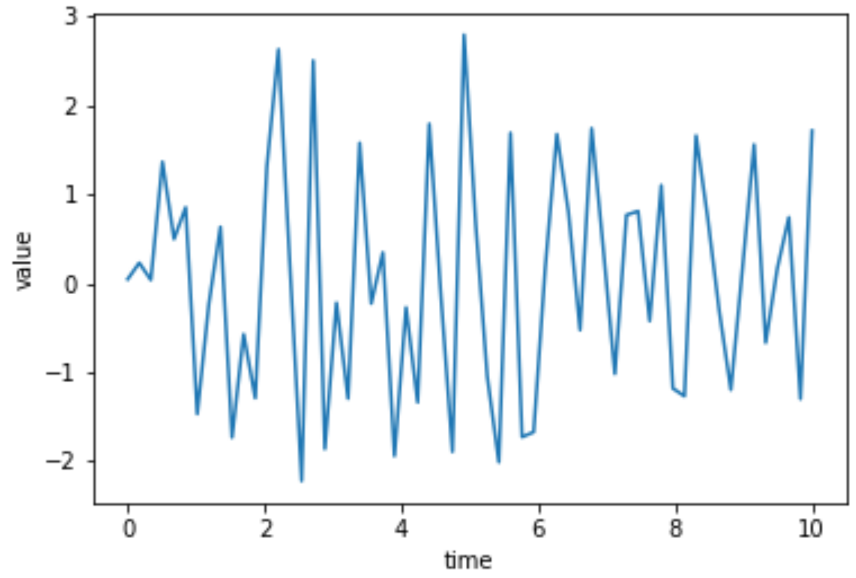
White noise

White noise series has uncorrelated values

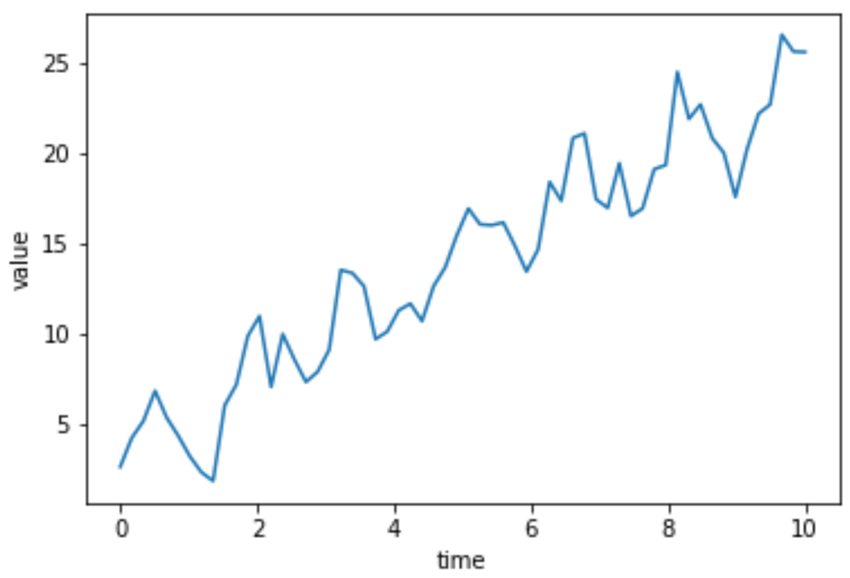
- Heads, heads, heads, tails, heads, tails, ...
- 0.1, -0.3, 0.8, 0.4, -0.5, 0.9, ...

Stationarity

Stationary



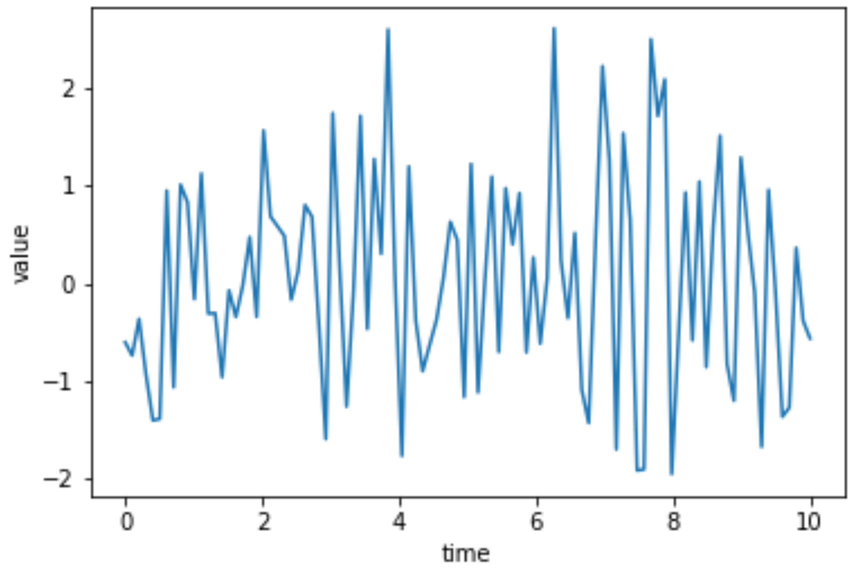
Not stationary



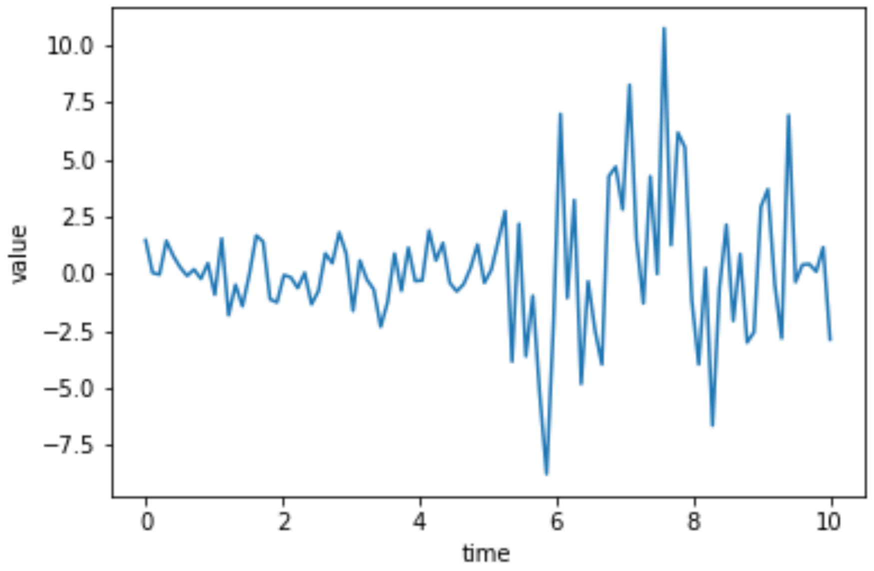
- Trend stationary: Trend is zero

Stationarity

Stationary



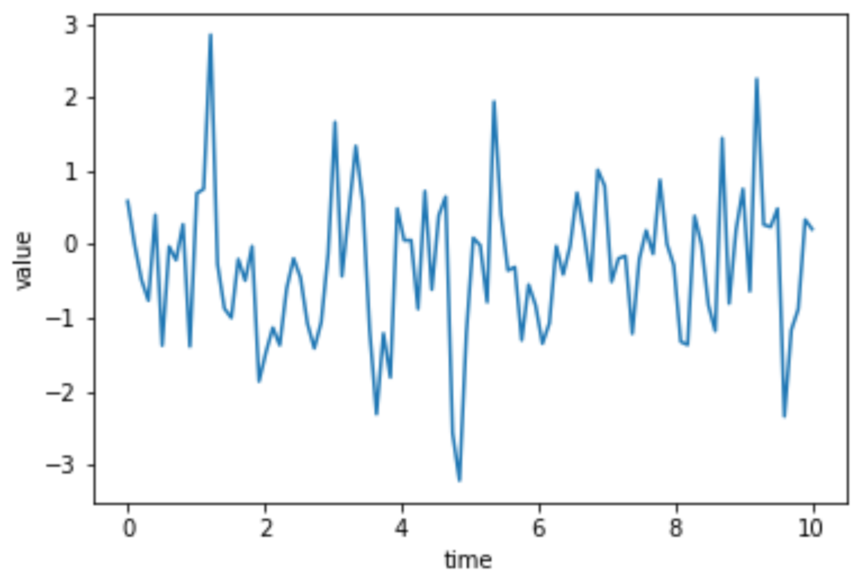
Not stationary



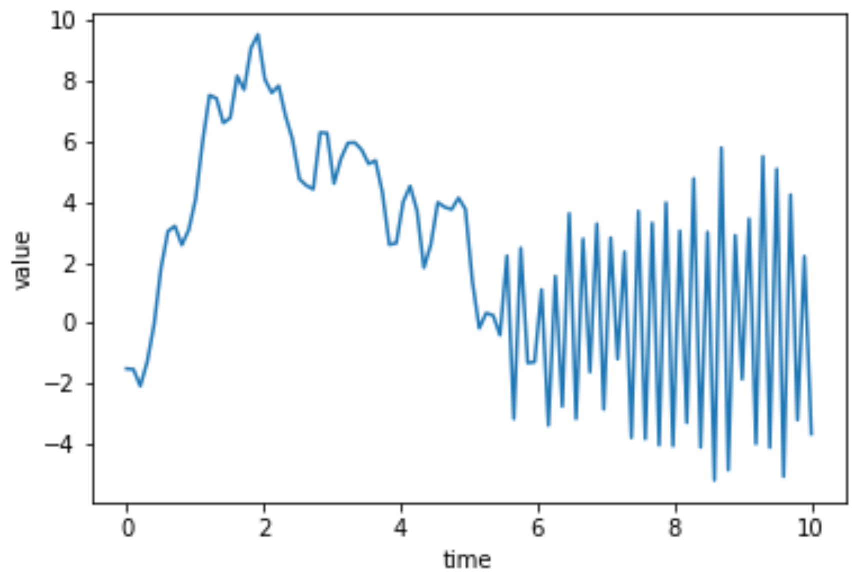
- Trend stationary: Trend is zero
- Variance is constant

Stationarity

Stationary



Not stationary



- Trend stationary: Trend is zero
- Variance is constant
- Autocorrelation is constant

Train-test split

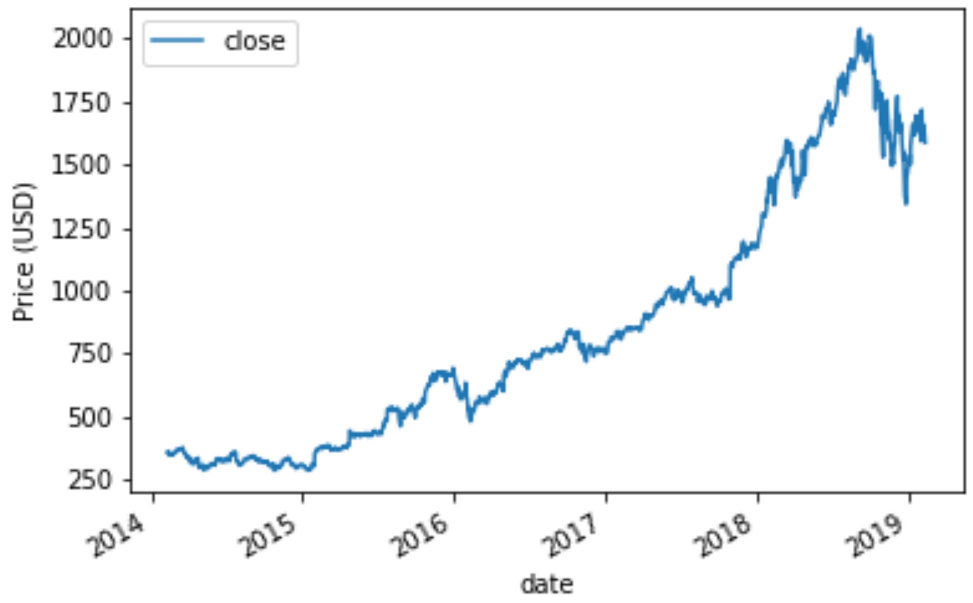
```
# Train data - all data up to the end of 2018
df_train = df.loc[:'2018']

# Test data - all data from 2019 onwards
df_test = df.loc['2019':]
```

The augmented Dicky-Fuller test

- Tests for trend non-stationarity
- Null hypothesis is time series is non-stationary

Applying the adfuller test




```
from statsmodels.tsa.stattools import adfuller

results = adfuller(df['close'])
```

Interpreting the test result

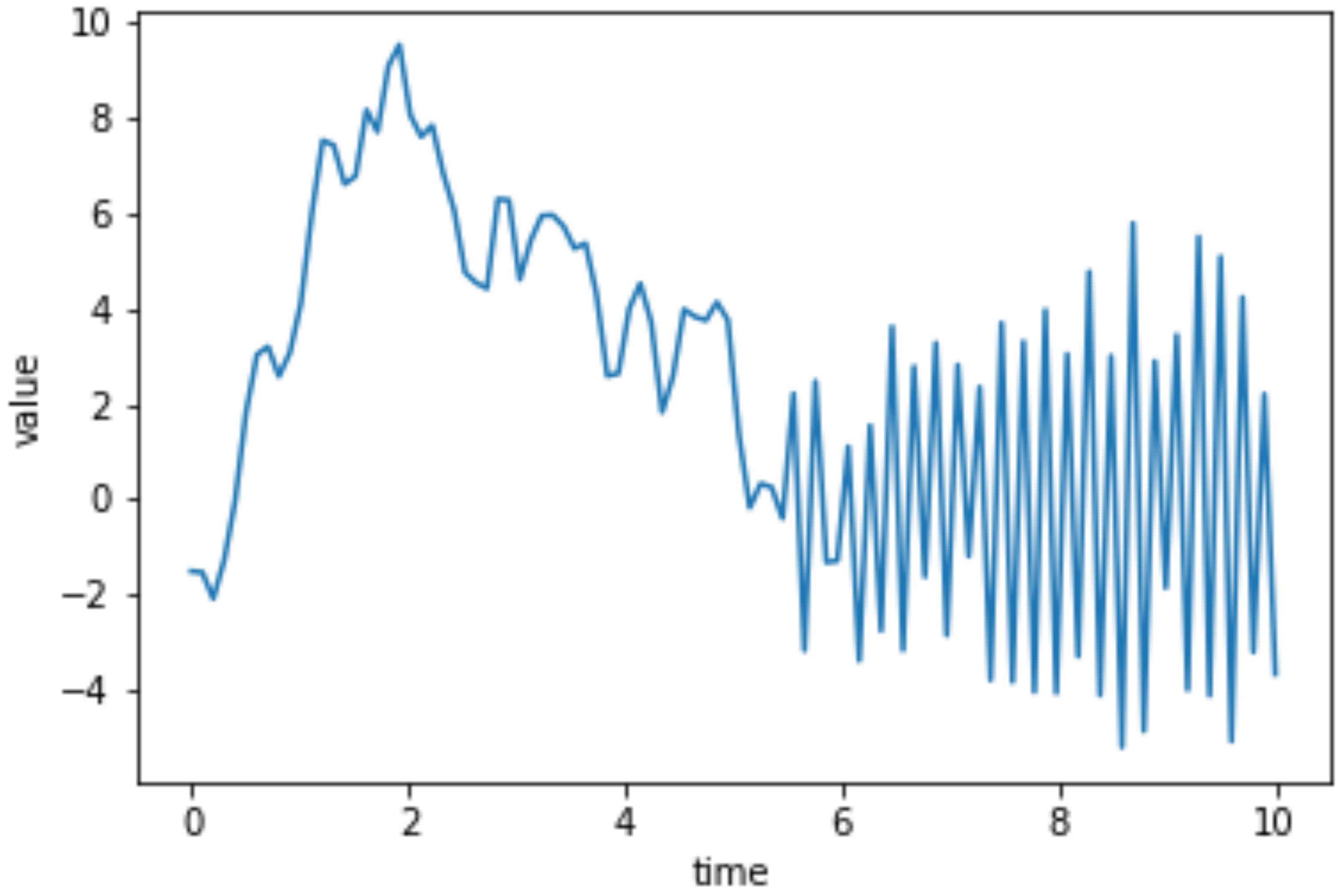
```
print(results)
```

```
(-1.34, 0.60, 23, 1235, {'1%': -3.435, '5%': -2.863, '10%': -2.568}, 10782.87)
```

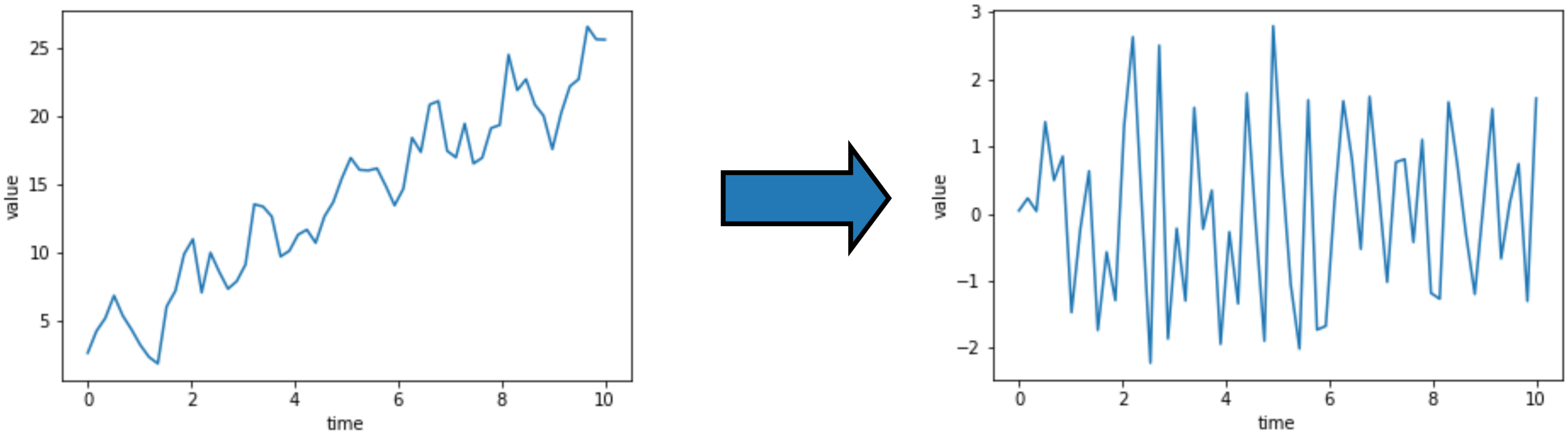
- 0th element is test statistic (-1.34)
  - More negative means more likely to be stationary
- 1st element is p-value: (0.60)
  - If p-value is small → reject null hypothesis. Reject non-stationary.
- 4th element is the critical test statistics

[1] https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html

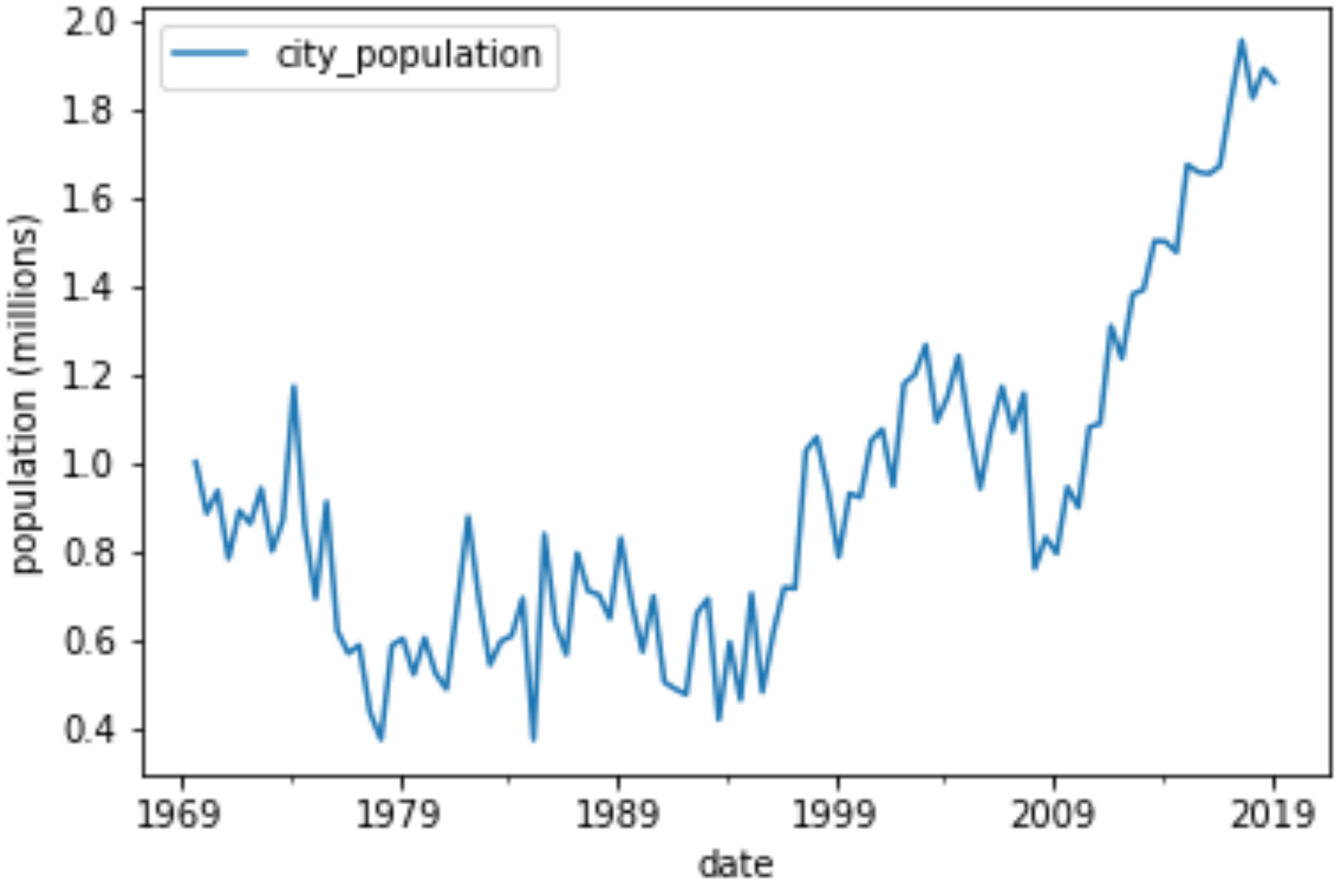
The value of plotting



Making a time series stationary



Taking the difference



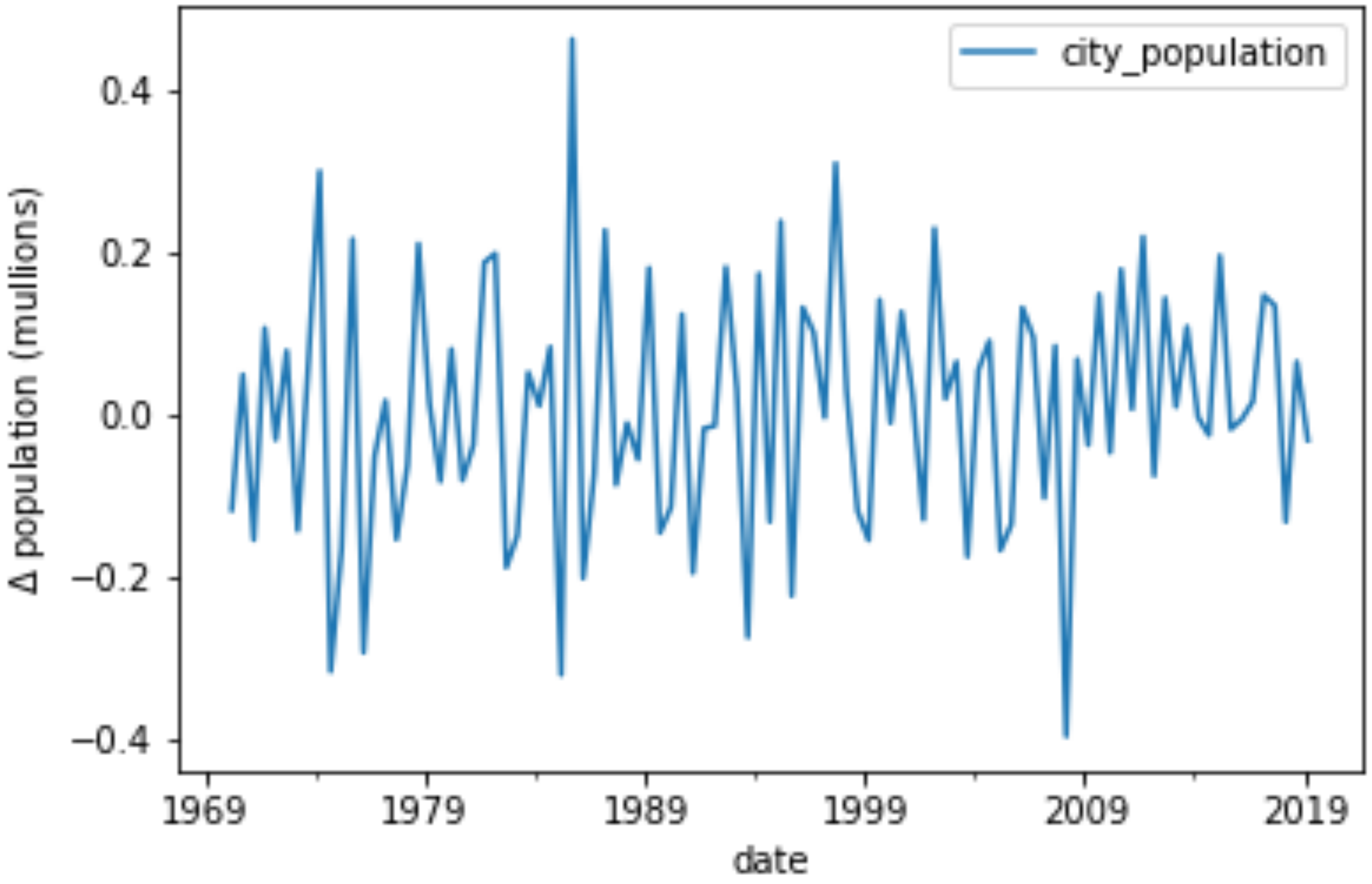
Difference: $\Delta y_t = y_t - y_{t-1}$

Taking the difference

```
df_stationary = df.diff().dropna()
```

```
            city_population
date
1970-03-31        -0.116156
1970-09-30         0.050850
1971-03-31        -0.153261
1971-09-30         0.108389
1972-03-31        -0.029569
```

Taking the difference



Other transforms

Examples of other transforms

- Take the log
  - `np.log(df)`
- Take the square root
  - `np.sqrt(df)`
- Take the proportional change
  - `df.shift(1)/df`

Intro to AR, MA and ARMA models

ARIMA MODELS IN PYTHON

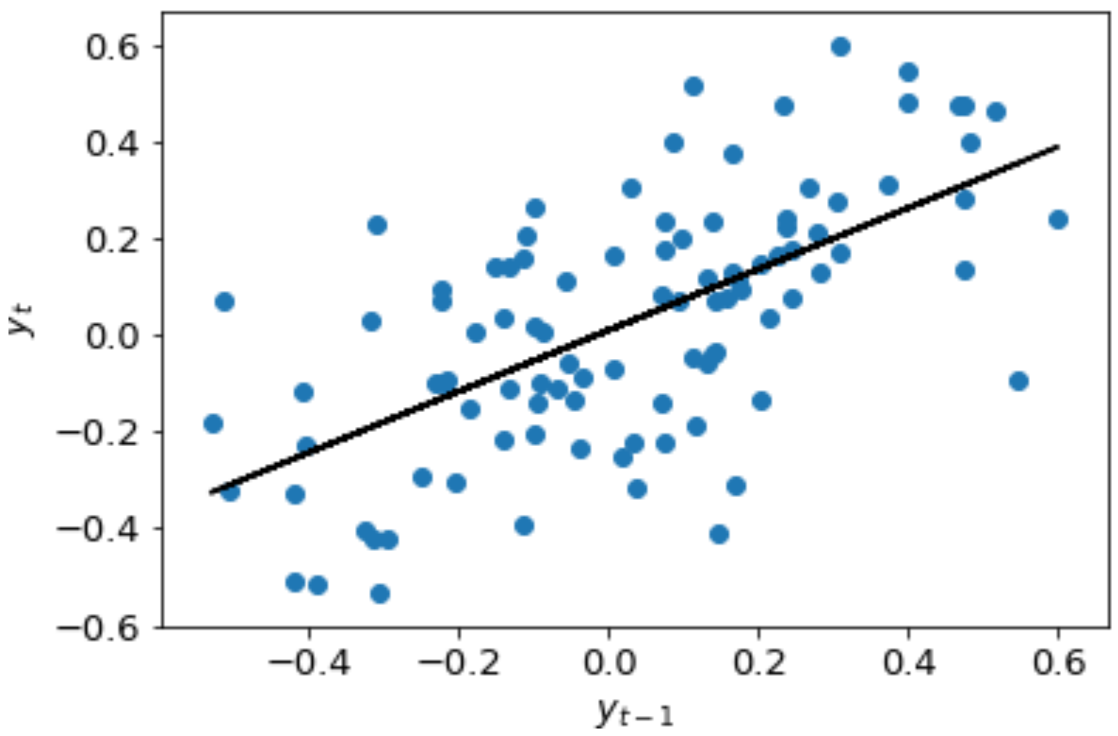
James Fulton
Climate informatics researcher

AR models

Autoregressive (AR) model

AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$



AR models

Autoregressive (AR) model

AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$

AR(2) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \epsilon_t$$

AR(p) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + ... + a_p y_{t-p} + \epsilon_t$$

MA models

Moving average (MA) model

MA(1) model :

$$y_t = m_1\epsilon_{t-1} + \epsilon_t$$

MA(2) model :

$$y_t = m_1\epsilon_{t-1} + m_2\epsilon_{t-2} + \epsilon_t$$

MA(q) model :

$$y_t = m_1\epsilon_{t-1} + m_2\epsilon_{t-2} + ... + m_q\epsilon_{t-q} + \epsilon_t$$

Fitting the model and fit summary

```
model = ARIMA(timeseries, order=(2,0,1))
results = model.fit()
```

```
print(results.summary())
```

ARMA models

Autoregressive moving-average (ARMA) model

- ARMA = AR + MA

ARMA(1,1) model :

$$y_t = a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

ARMA(p, q)

- p is order of AR part
- q is order of MA part

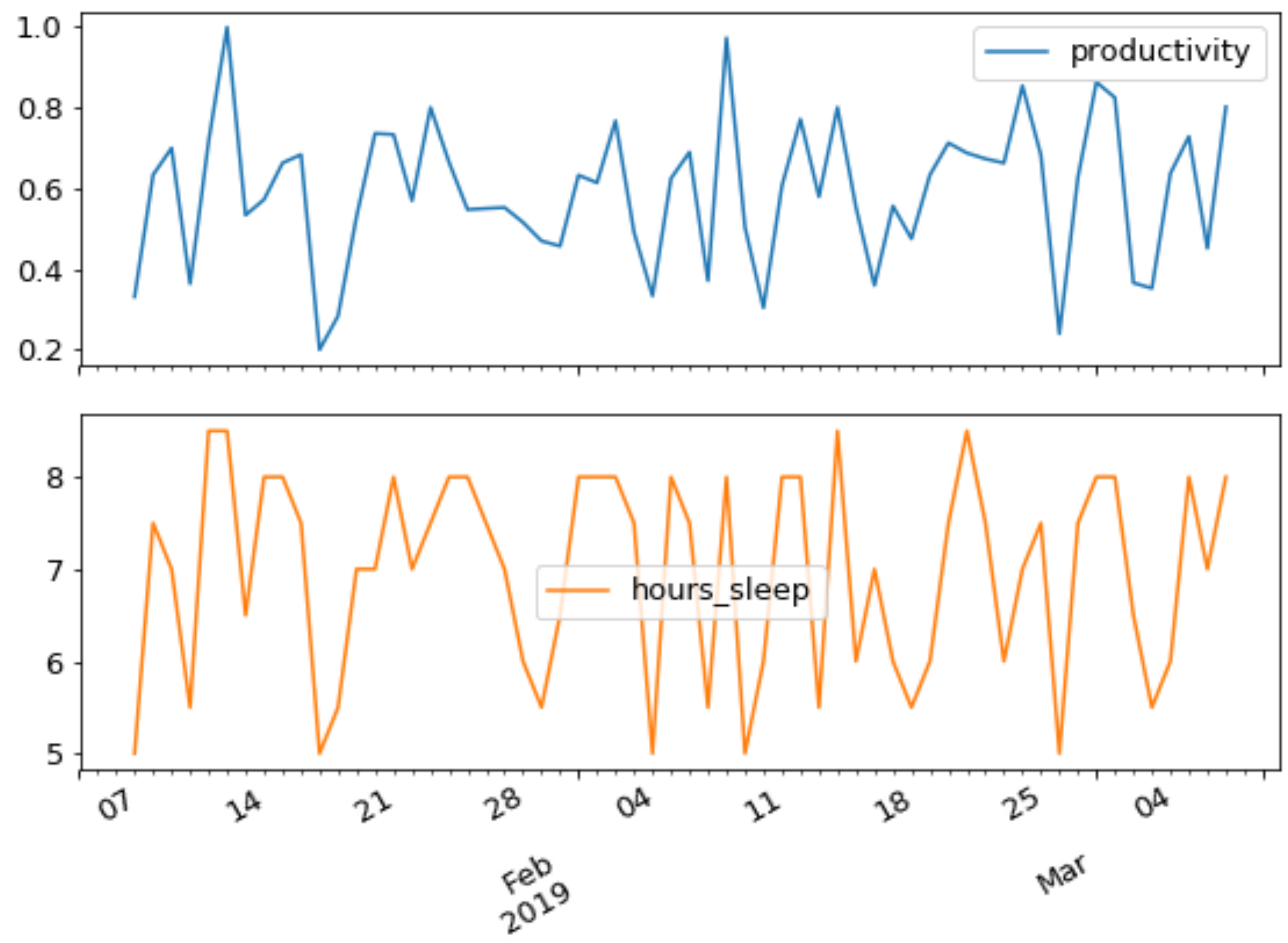
Fit summary

```
                           SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1000
Model:                     ARMA(2, 1)   Log Likelihood                 148.580
Date:                Thu, 25 Apr 2022   AIC                           -287.159
Time:                        22:57:00   BIC                           -262.621
Sample:                             0   HQIC                          -277.833
Covariance Type:                  opg
==============================================================================
                coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -0.0017      0.012     -0.147      0.883      -0.025       0.021
ar.L1.y       0.5253      0.054      9.807      0.000       0.420       0.630
ar.L2.y      -0.2909      0.042     -6.850      0.000      -0.374      -0.208
ma.L1.y       0.3679      0.052      7.100      0.000       0.266       0.469
```

Introduction to ARMAX models

- Exogenous ARMA
- Use external variables as well as time series
- ARMAX = ARMA + linear regression

ARMAX example



Fitting ARMAX

```
# Instantiate the model
model = ARIMA(df['productivity'], order=(2,0,1), exog=df['hours_sleep'])


# Fit the model
results = model.fit()
```

ARMAX summary

```
==================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const         -0.1936      0.092     -2.098      0.041      -0.375      -0.013
x1             0.1131      0.013      8.602      0.000       0.087       0.139
ar.L1.y        0.1917      0.252      0.760      0.450      -0.302       0.686
ar.L2.y       -0.3740      0.121     -3.079      0.003      -0.612      -0.136
ma.L1.y       -0.0740      0.259     -0.286      0.776      -0.581       0.433
```

Predicting the next value

Take an AR(1) model

$$y_t = a_1 y_{t-1} + \epsilon_t$$

Predict next value

$$y_t = 0.6 \text{ x } 10 + \epsilon_t$$

$$y_t = 6.0 + \epsilon_t$$

Uncertainty on prediction

$$5.0 < y_t < 7.0$$

Making one-step-ahead predictions

```
# Make predictions for last 25 values
results = model.fit()
# Make in-sample prediction
forecast = results.get_prediction(start=-25)
# forecast mean
mean_forecast = forecast.predicted_mean
```

Predicted mean is a pandas series

```
2013-10-28    1.519368
2013-10-29    1.351082
2013-10-30    1.218016
```

Confidence intervals

```
# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

Confidence interval method returns `pandas` DataFrame

```
                              lower y                      upper y
2013-09-28                  -4.720471                    -0.815384
2013-09-29                  -5.069875                     0.112505
2013-09-30                  -5.232837                     0.766300
2013-10-01                  -5.305814                     1.282935
2013-10-02                  -5.326956                     1.703974
```

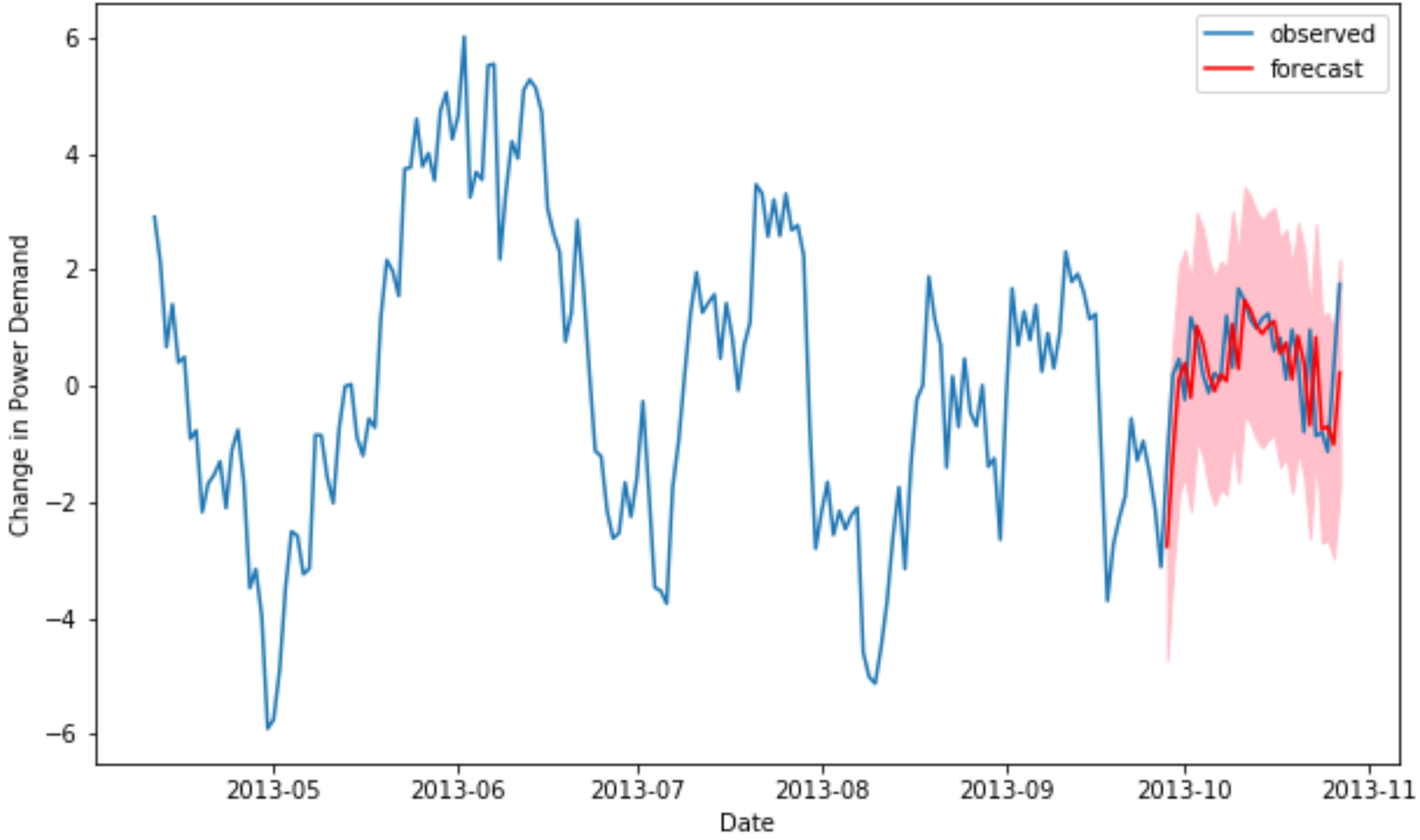
Plotting predictions

```
plt.figure()

# Plot prediction
plt.plot(dates,
         mean_forecast.values,
         color='red',
         label='forecast')
# Shade uncertainty area
plt.fill_between(dates, lower_limits, upper_limits, color='pink')

plt.show()
```

Plotting predictions



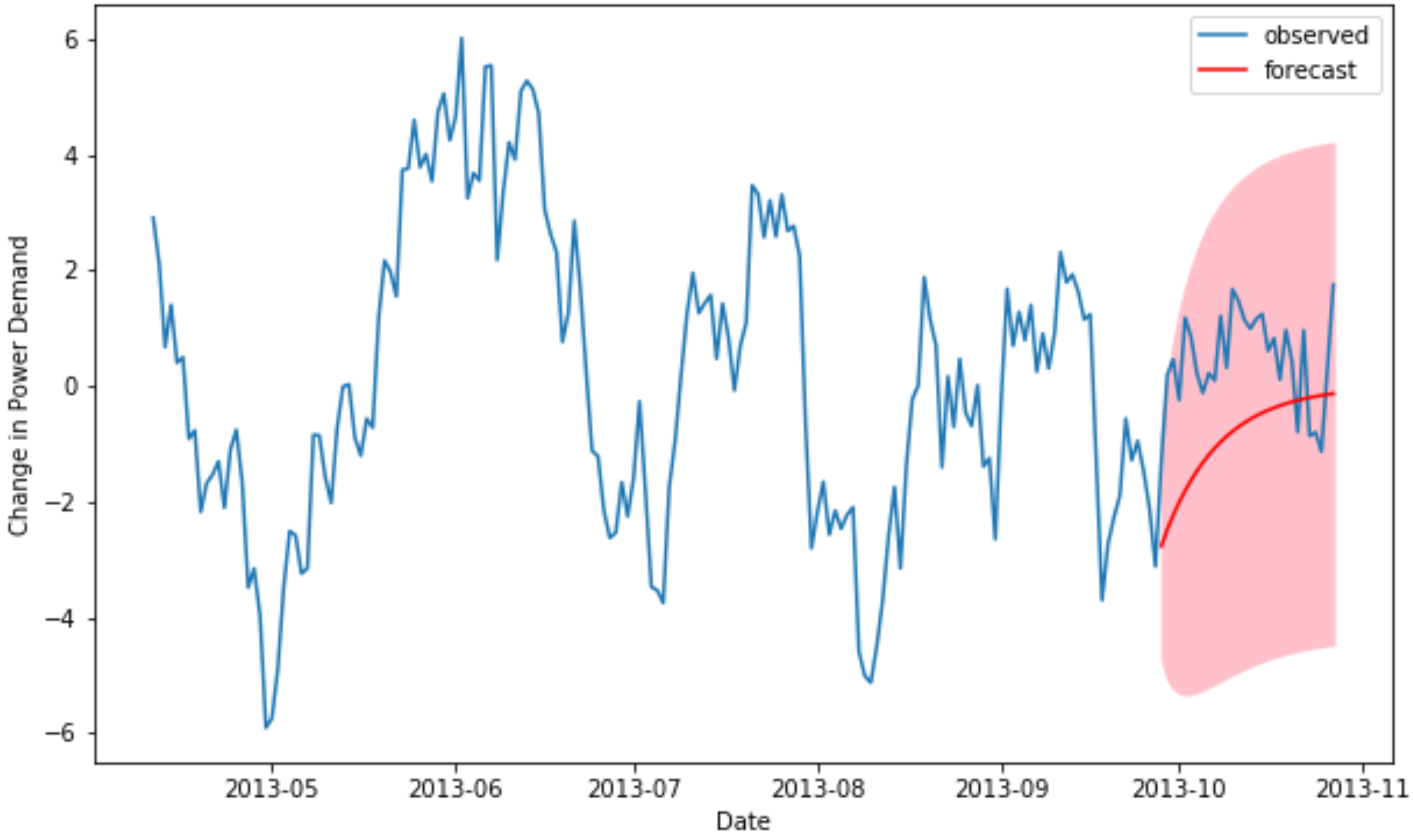
Making dynamic predictions

```
results = model.fit()
forecast = results.get_prediction(start=-25, dynamic=True)
```

```
# forecast mean
mean_forecast = forecast.predicted_mean

# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

Dynamic predictions



Forecasting out of sample

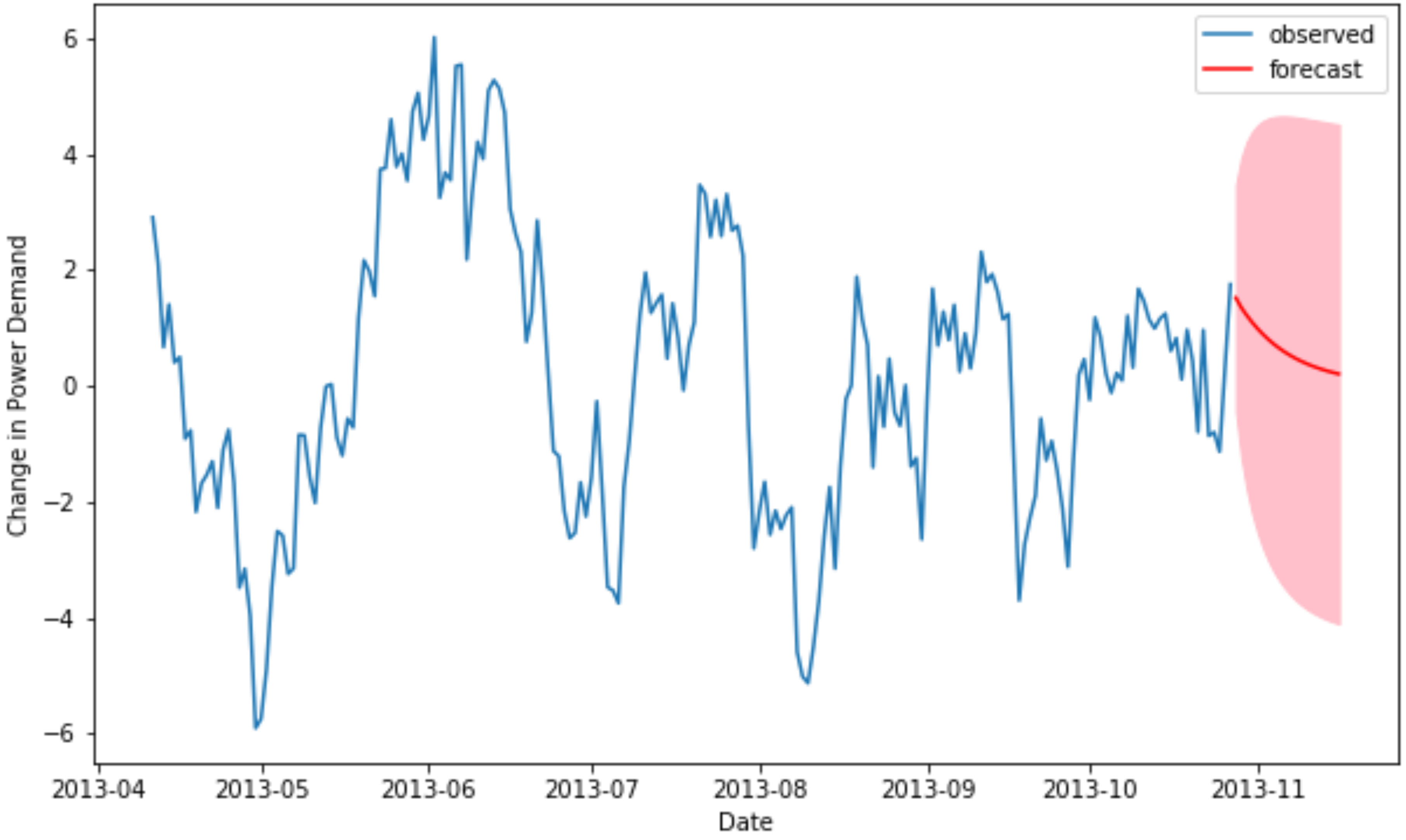
```
forecast = results.get_forecast(steps=20)
```

```
# forecast mean
mean_forecast = forecast.predicted_mean

# Get confidence intervals of forecasts
confidence_intervals = forecast.conf_int()
```

Forecasting out of sample

```
forecast = results.get_forecast(steps=20)
```



Introduction to ARIMA models

ARIMA MODELS IN PYTHON

James Fulton
Climate informatics researcher

The ARIMA model

- Take the difference
- Fit ARMA model
- Integrate forecast

Can we avoid doing so much work?

Yes!

ARIMA - Autoregressive Integrated Moving Average

Using the ARIMA model

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df, order=(p,d,q))
```

- p - number of autoregressive lags
- d - order of differencing
- q - number of moving average lags

ARIMA$(p, 0, q)$ = ARMA(p, q)

Picking the difference order

```
adf = adfuller(df.iloc[:,0])
print('ADF Statistic:', adf[0])
print('p-value:', adf[1])
```

```
ADF Statistic: -2.674
p-value: 0.0784
```

```
adf = adfuller(df.diff().dropna().iloc[:,0])
print('ADF Statistic:', adf[0])
print('p-value:', adf[1])
```

```
ADF Statistic: -4.978
p-value: 2.44e-05
```

Using the ARIMA model

```
# Create model
model = ARIMA(df, order=(2,1,1))
# Fit model
model.fit()
# Make forecast
mean_forecast = results.get_forecast(steps=10).predicted_mean
```

Using the ARIMA model

```
# Make forecast
mean_forecast = results.get_forecast(steps=steps).predicted_mean
```